Student name: Lê Ngọc Anh Vũ

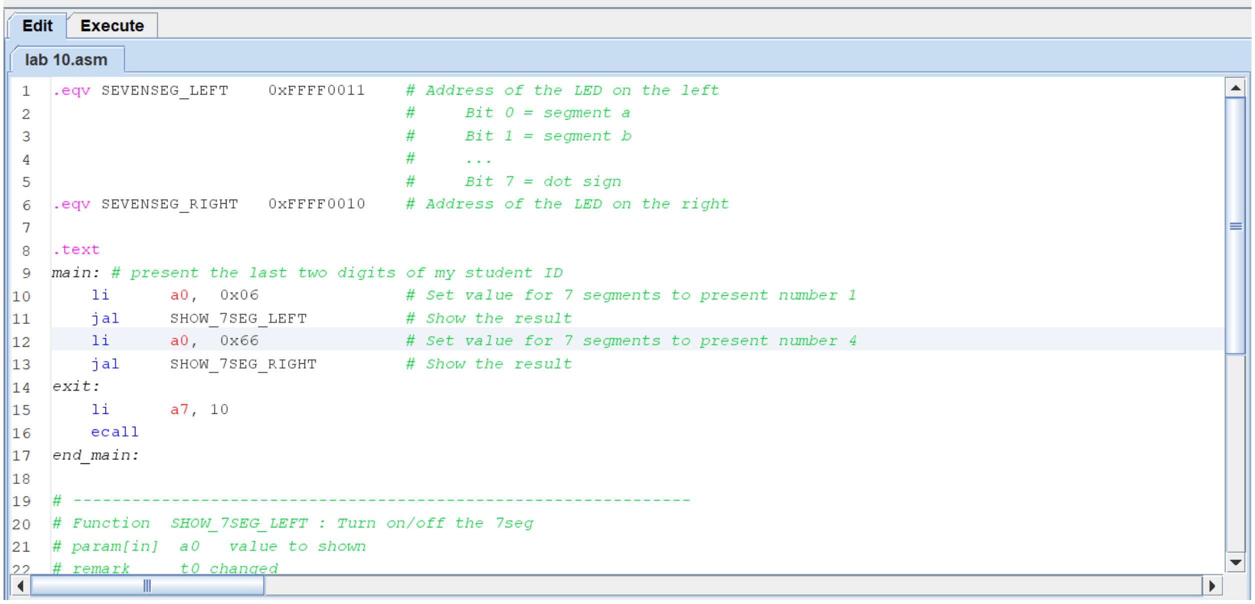
Student ID: 20236014

LAB 10

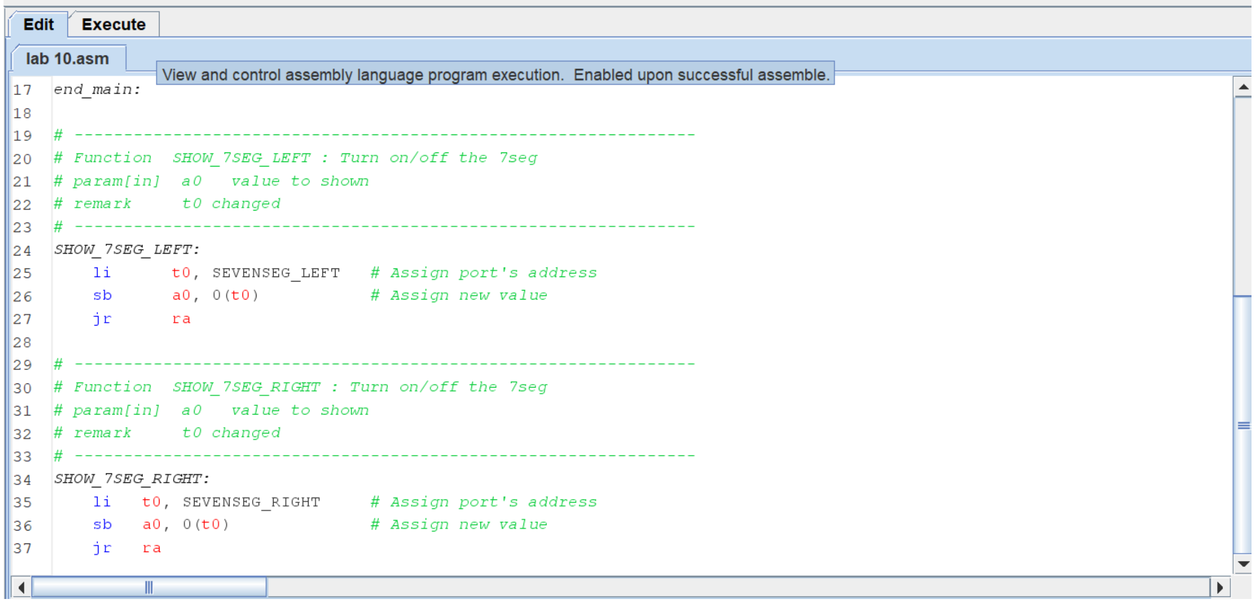
Assignment 1: Implement the program in Home Assignment 1, change the values displayed on the LEDs such as the last two digits of StudentID and the last two digits of the ASCII code of a character entered from the keyboard.

- The last two digits of my StudentID are presented on the LEDs:

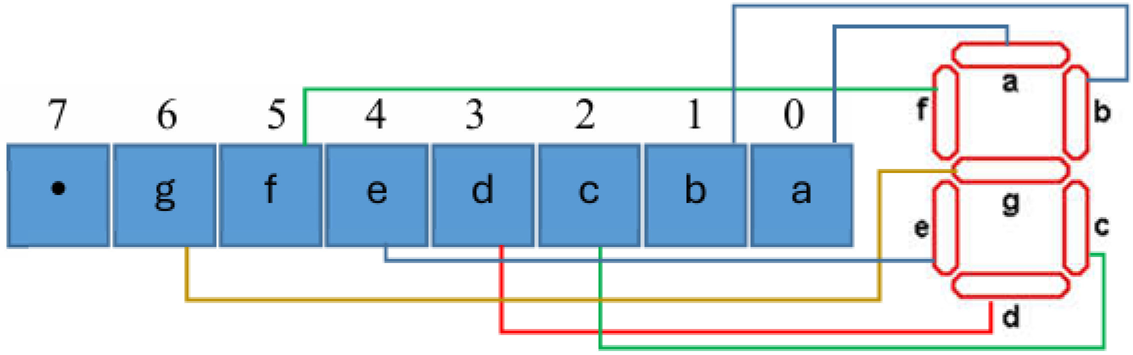
My studentID is 20236014. It means that the LEDs have to display number 14.



```
.eqv SEVENSEG_LEFT     0xFFFF0011     # Address of the LED on the left
                                      #     Bit 0 = segment a
                                      #     Bit 1 = segment b
                                      #     ...
                                      #     Bit 7 = dot sign
.eqv SEVENSEG_RIGHT    0xFFFF0010     # Address of the LED on the right

.text
main: # present the last two digits of my student ID
    li      a0,  0x06                 # Set value for 7 segments to present number 1
    jal     SHOW_7SEG_LEFT            # Show the result
    li      a0,  0x66                 # Set value for 7 segments to present number 4
    jal     SHOW_7SEG_RIGHT           # Show the result
exit:
    li      a7, 10
    ecall
end_main:

# ---------------------------------------------------------------
# Function  SHOW_7SEG_LEFT : Turn on/off the 7seg
# param[in]  a0   value to shown
# remark      t0 changed
```


Edit
Execute
lab 10.asm
View and control assembly language program execution. Enabled upon successful assemble.

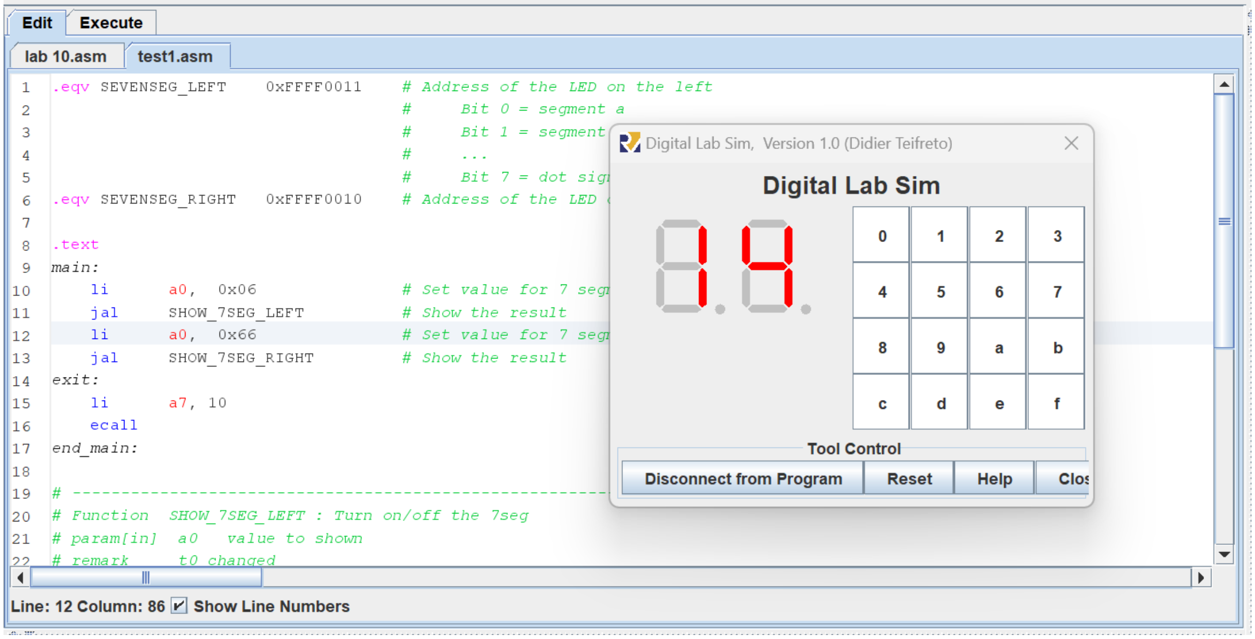
```
end_main:

# ---------------------------------------------------------------
# Function  SHOW_7SEG_LEFT : Turn on/off the 7seg
# param[in]  a0   value to shown
# remark      t0 changed
# ---------------------------------------------------------------
SHOW_7SEG_LEFT:
    li      t0, SEVENSEG_LEFT    # Assign port's address
    sb      a0, 0(t0)            # Assign new value
    jr      ra

# ---------------------------------------------------------------
# Function  SHOW_7SEG_RIGHT : Turn on/off the 7seg
# param[in]  a0   value to shown
# remark      t0 changed
# ---------------------------------------------------------------
SHOW_7SEG_RIGHT:
    li   t0, SEVENSEG_RIGHT      # Assign port's address
    sb   a0, 0(t0)               # Assign new value
    jr   ra
```



The right seven segment display has to present number 4. Then, bits in segment b, c, f, g are bit 1s. Hence, the right seven segment display receives value of 01100110 in binary or 0x66 in hexadecimal.

Similarly, the left seven segment display has to present number 1. Then, bits in segment b, c are bit 1s. Hence, the left seven segment display gets value of 00000110 in binary or 0x06 in hexadecimal.

Finally, the result is 14

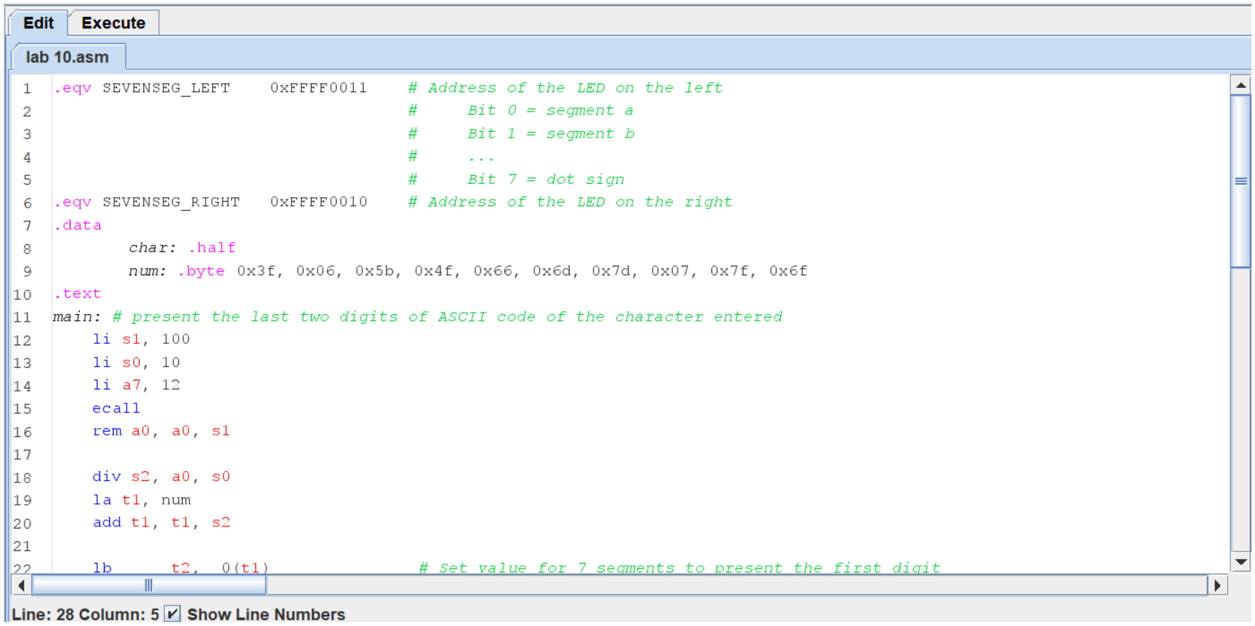


```
.eqv SEVENSEG_LEFT    0xFFFF0011     # Address of the LED on the left
                                     #     Bit 0 = segment a
                                     #     Bit 1 = segment
                                     #     ...
                                     #     Bit 7 = dot sig
.eqv SEVENSEG_RIGHT   0xFFFF0010     # Address of the LED

.text
main:
    li      a0,  0x06                # Set value for 7 seg
    jal     SHOW_7SEG_LEFT           # Show the result
    li      a0,  0x66                # Set value for 7 seg
    jal     SHOW_7SEG_RIGHT          # Show the result
exit:
    li      a7, 10
    ecall
end_main:

# ---------------------------------------------------------
# Function  SHOW_7SEG_LEFT : Turn on/off the 7seg
# param[in]  a0   value to shown
# remark     t0 changed
```

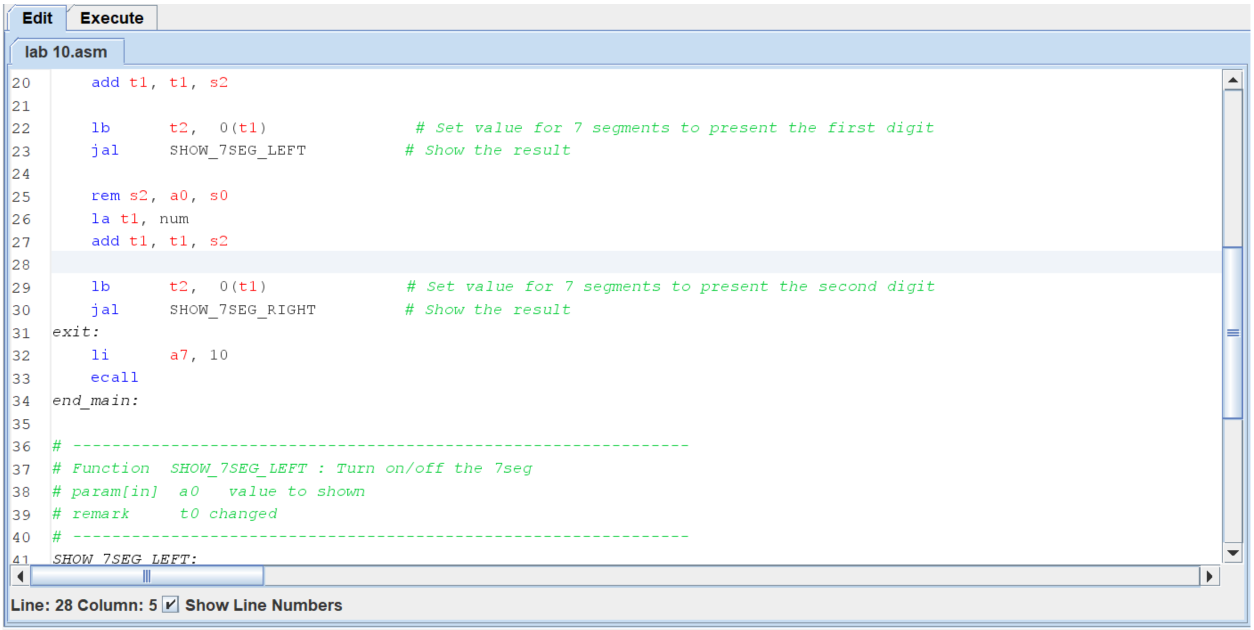
Line: 12 Column: 86 ☑ Show Line Numbers

- The last two digits of the ASCII code of a character entered from the keyboard.

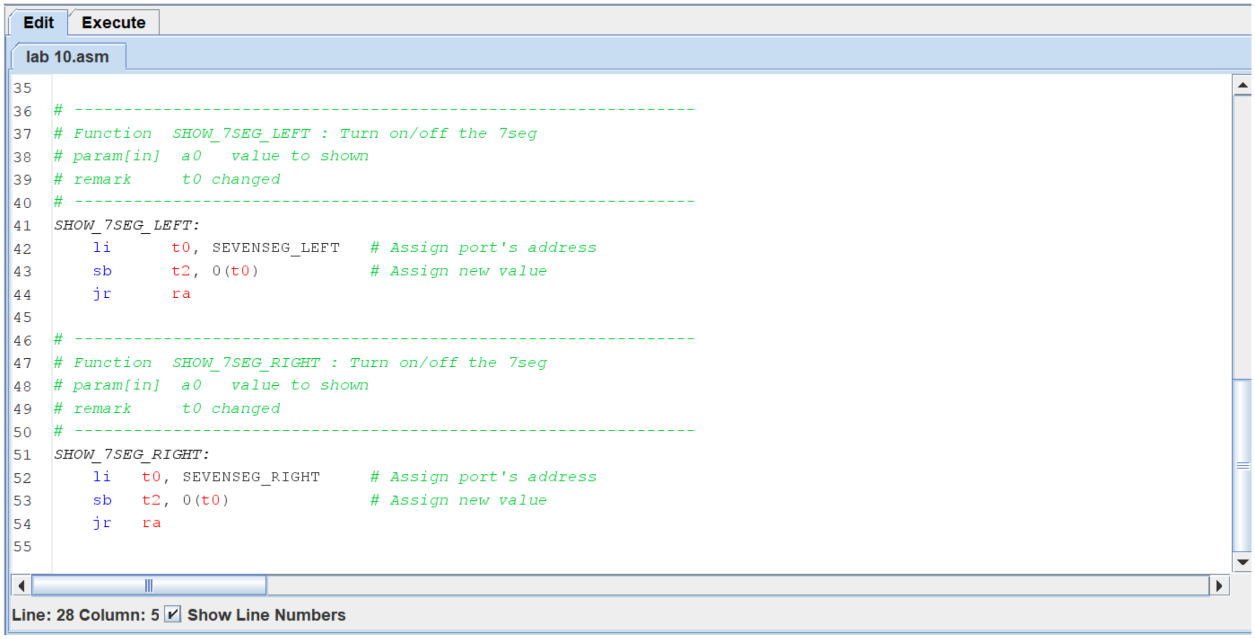
The source code is as below:



```
.eqv SEVENSEG_LEFT     0xFFFF0011     # Address of the LED on the left
                                      #     Bit 0 = segment a
                                      #     Bit 1 = segment b
                                      #     ...
                                      #     Bit 7 = dot sign
.eqv SEVENSEG_RIGHT    0xFFFF0010     # Address of the LED on the right
.data
        char: .half
        num: .byte 0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f
.text
main: # present the last two digits of ASCII code of the character entered
    li s1, 100
    li s0, 10
    li a7, 12
    ecall
    rem a0, a0, s1

    div s2, a0, s0
    la t1, num
    add t1, t1, s2

    lb      t2,  0(t1)                 # Set value for 7 segments to present the first digit
```



```
    add t1, t1, s2

    lb      t2,  0(t1)                 # Set value for 7 segments to present the first digit
    jal     SHOW_7SEG_LEFT            # Show the result

    rem s2, a0, s0
    la t1, num
    add t1, t1, s2

    lb      t2,  0(t1)                # Set value for 7 segments to present the second digit
    jal     SHOW_7SEG_RIGHT           # Show the result
exit:
    li      a7, 10
    ecall
end_main:

# ---------------------------------------------------------------
# Function  SHOW_7SEG_LEFT : Turn on/off the 7seg
# param[in]  a0   value to shown
# remark     t0 changed
# ---------------------------------------------------------------
SHOW_7SEG_LEFT:
```



```

# ----------------------------------------------------------------
# Function  SHOW_7SEG_LEFT : Turn on/off the 7seg
# param[in]  a0   value to shown
# remark     t0 changed
# ----------------------------------------------------------------
SHOW_7SEG_LEFT:
    li      t0, SEVENSEG_LEFT    # Assign port's address
    sb      t2, 0(t0)            # Assign new value
    jr      ra

# ----------------------------------------------------------------
# Function  SHOW_7SEG_RIGHT : Turn on/off the 7seg
# param[in]  a0   value to shown
# remark     t0 changed
# ----------------------------------------------------------------
SHOW_7SEG_RIGHT:
    li   t0, SEVENSEG_RIGHT      # Assign port's address
    sb   t2, 0(t0)               # Assign new value
    jr   ra

```

Line: 28 Column: 5 ☑ Show Line Numbers

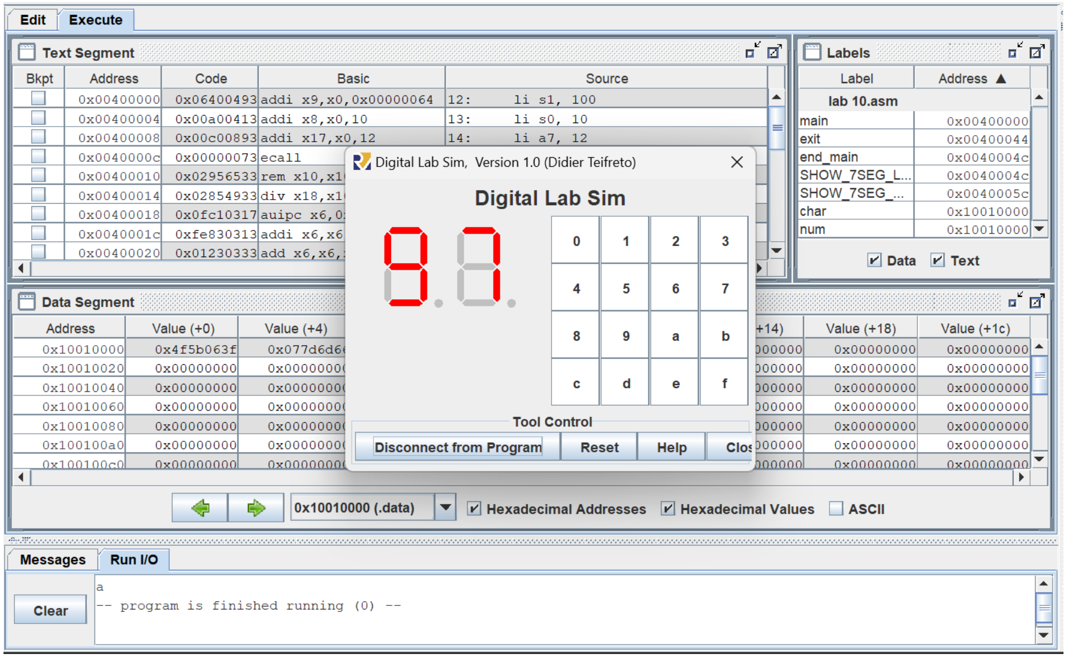
I create an array num containing the value of seven segment display of the digit from 0 to 9. Num[i] will display the digit i. Then, consider the last second digit and the last digit. Register t1 stores address of array num.

⇨ (t1 + i) stores the value of seven segment display to display digit i.

Hence, we can easily present the last two digits of the ASCII code of a character entered from the keyboard.

For some examples:

- If we input ‘a’, the result is:



Bkpt	Address	Code	Basic	Source
□	0x00400000	0x06400493	addi x9,x0,0x00000064	12: li s1, 100
□	0x00400004	0x00a00413	addi x8,x0,10	13: li s0, 10
□	0x00400008	0x00c00893	addi x17,x0,12	14: li a7, 12
□	0x0040000c	0x00000073	ecall	
□	0x00400010	0x02956533	rem x10,x1	
□	0x00400014	0x02854933	div x18,x1	
□	0x00400018	0x0fc10317	auipc x6,0	
□	0x0040001c	0xfe830313	addi x6,x6	
□	0x00400020	0x01230333	add x6,x6,	

Label	Address ▲
lab 10.asm	
main	0x00400000
exit	0x00400044
end_main	0x0040004c
SHOW_7SEG_L...	0x0040004c
SHOW_7SEG_...	0x0040005c
char	0x10010000
num	0x10010000

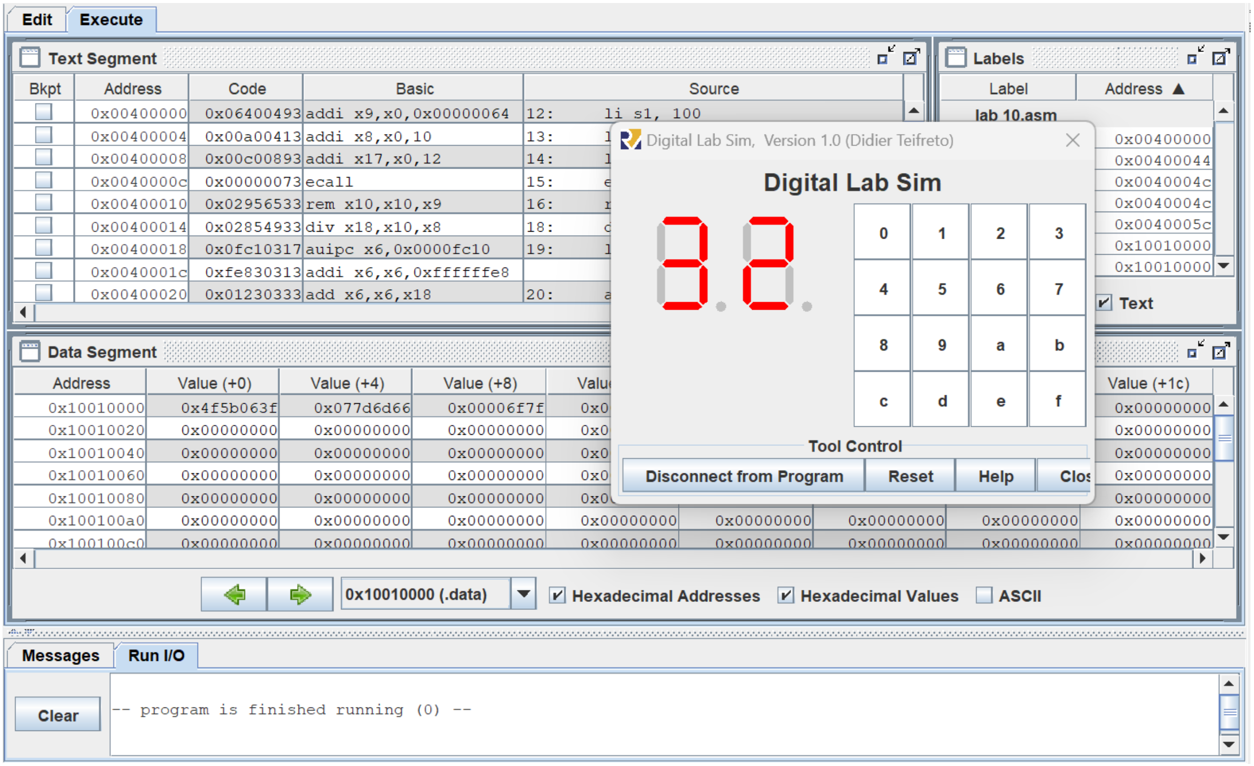
☑ Data ☑ Text

Address	Value (+0)	Value (+4)	+14)	Value (+18)	Value (+1c)
0x10010000	0x4f5b063f	0x077d6d6	000000	0x00000000	0x00000000
0x10010020	0x00000000	0x0000000	000000	0x00000000	0x00000000
0x10010040	0x00000000	0x0000000	000000	0x00000000	0x00000000
0x10010060	0x00000000	0x0000000	000000	0x00000000	0x00000000
0x10010080	0x00000000	0x0000000	000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x0000000	000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x0000000	000000	0x00000000	0x00000000

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values □ ASCII

```
a
-- program is finished running (0) --
```

- If we input space, the result is:



Bkpt	Address	Code	Basic	Source
□	0x00400000	0x06400493	addi x9,x0,0x00000064	12: li s1, 100
□	0x00400004	0x00a00413	addi x8,x0,10	13: l
□	0x00400008	0x00c00893	addi x17,x0,12	14: l
□	0x0040000c	0x00000073	ecall	15: e
□	0x00400010	0x02956533	rem x10,x10,x9	16: r
□	0x00400014	0x02854933	div x18,x10,x8	18: d
□	0x00400018	0x0fc10317	auipc x6,0x0000fc10	19: l
□	0x0040001c	0xfe830313	addi x6,x6,0xffffffe8	
□	0x00400020	0x01230333	add x6,x6,x18	20: a

Label	Address ▲
lab 10.asm	
	0x00400000
	0x00400044
	0x0040004c
	0x0040004c
	0x0040005c
	0x10010000
	0x10010000

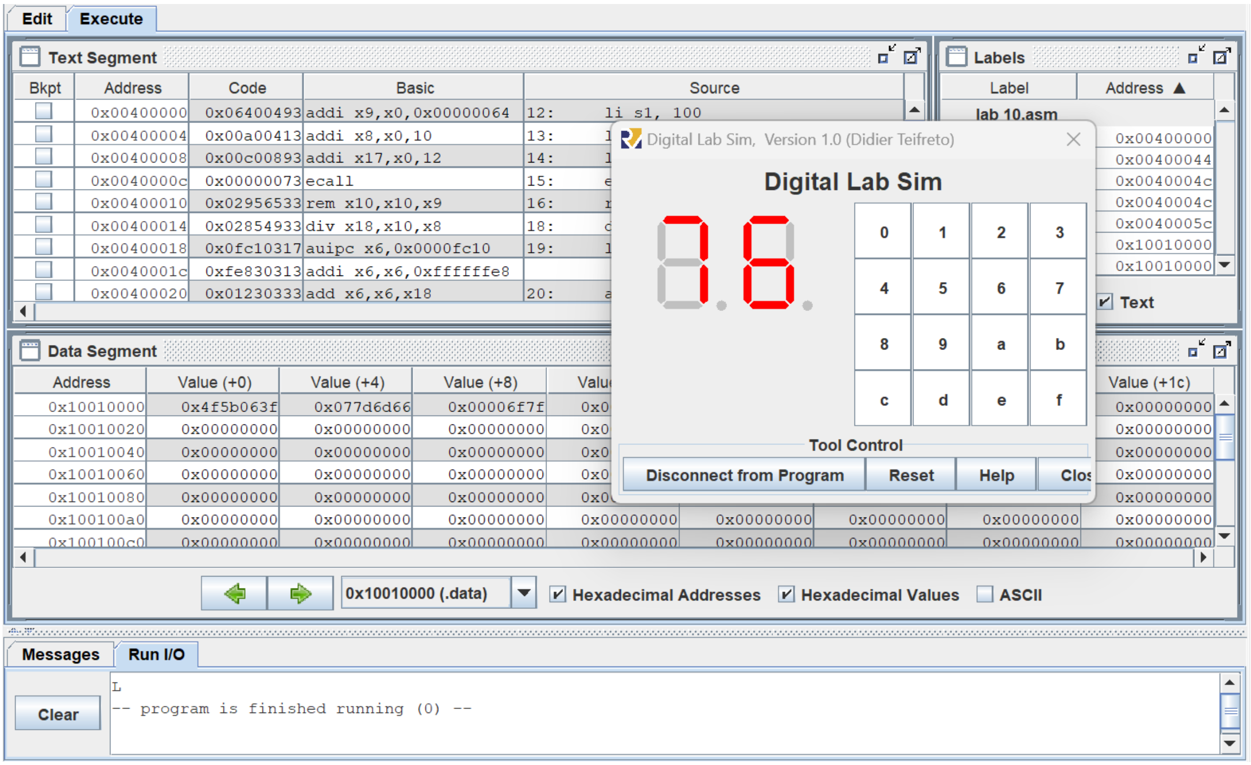
☑ Text

Address	Value (+0)	Value (+4)	Value (+8)	Valu				Value (+1c)
0x10010000	0x4f5b063f	0x077d6d66	0x00006f7f	0x0				0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values □ ASCII

```
-- program is finished running (0) --
```

- If we input ‘L’ , the result is:



Bkpt	Address	Code	Basic	Source
□	0x00400000	0x06400493	addi x9,x0,0x00000064	12: li s1, 100
□	0x00400004	0x00a00413	addi x8,x0,10	13: l
□	0x00400008	0x00c00893	addi x17,x0,12	14: l
□	0x0040000c	0x00000073	ecall	15: e
□	0x00400010	0x02956533	rem x10,x10,x9	16: r
□	0x00400014	0x02854933	div x18,x10,x8	18: d
□	0x00400018	0x0fc10317	auipc x6,0x0000fc10	19: l
□	0x0040001c	0xfe830313	addi x6,x6,0xffffffe8	
□	0x00400020	0x01230333	add x6,x6,x18	20: a

Label	Address ▲
lab 10.asm	
	0x00400000
	0x00400044
	0x0040004c
	0x0040004c
	0x0040005c
	0x10010000
	0x10010000

☑ Text

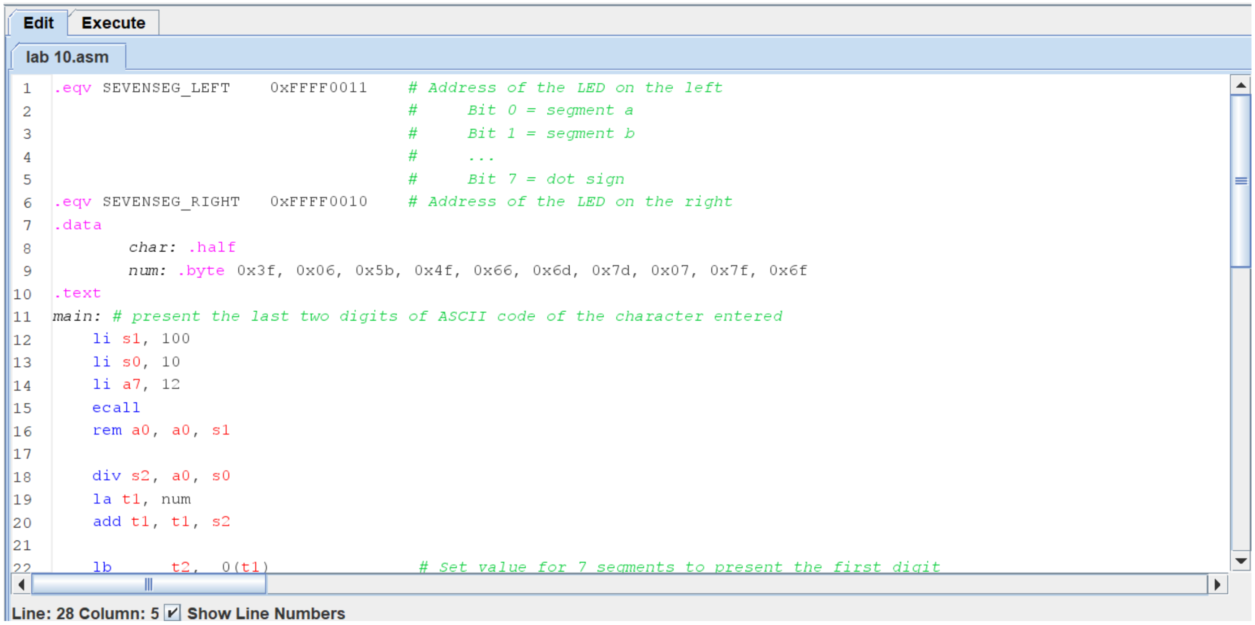
Address	Value (+0)	Value (+4)	Value (+8)	Valu				Value (+1c)
0x10010000	0x4f5b063f	0x077d6d66	0x00006f7f	0x0				0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values □ ASCII

```
L
-- program is finished running (0) --
```

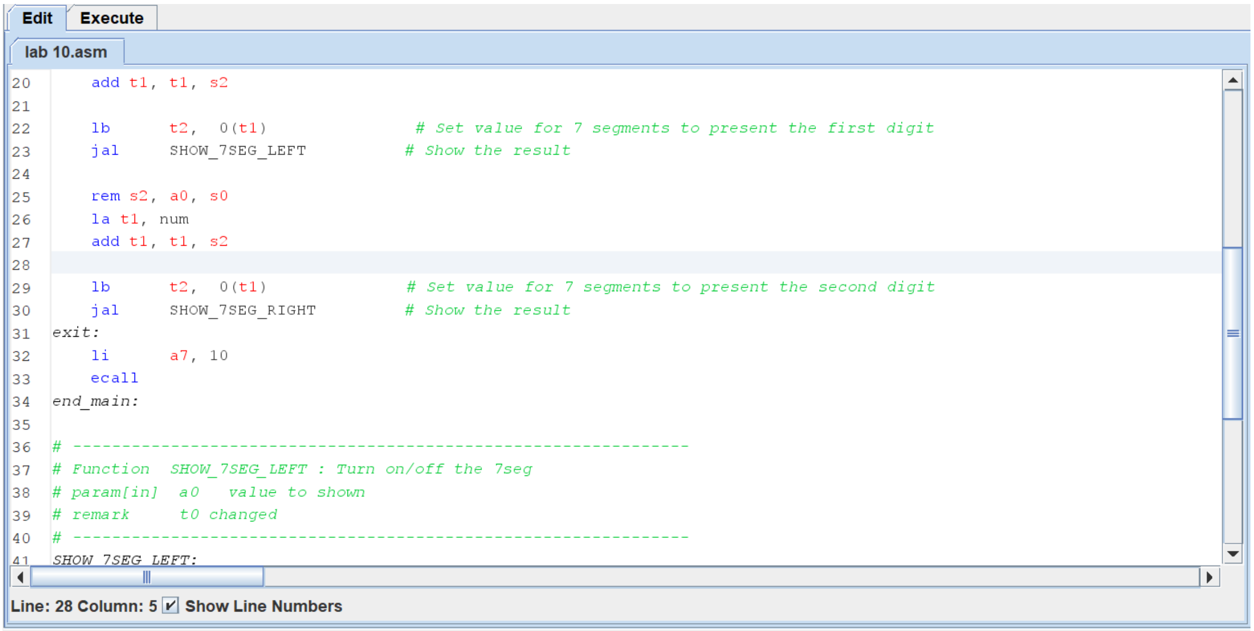
Assignment 2: Write a program that lets user enter a character from the keyboard and the program will print the last two digits of the ASCII code of the characters.

The source code is as below:



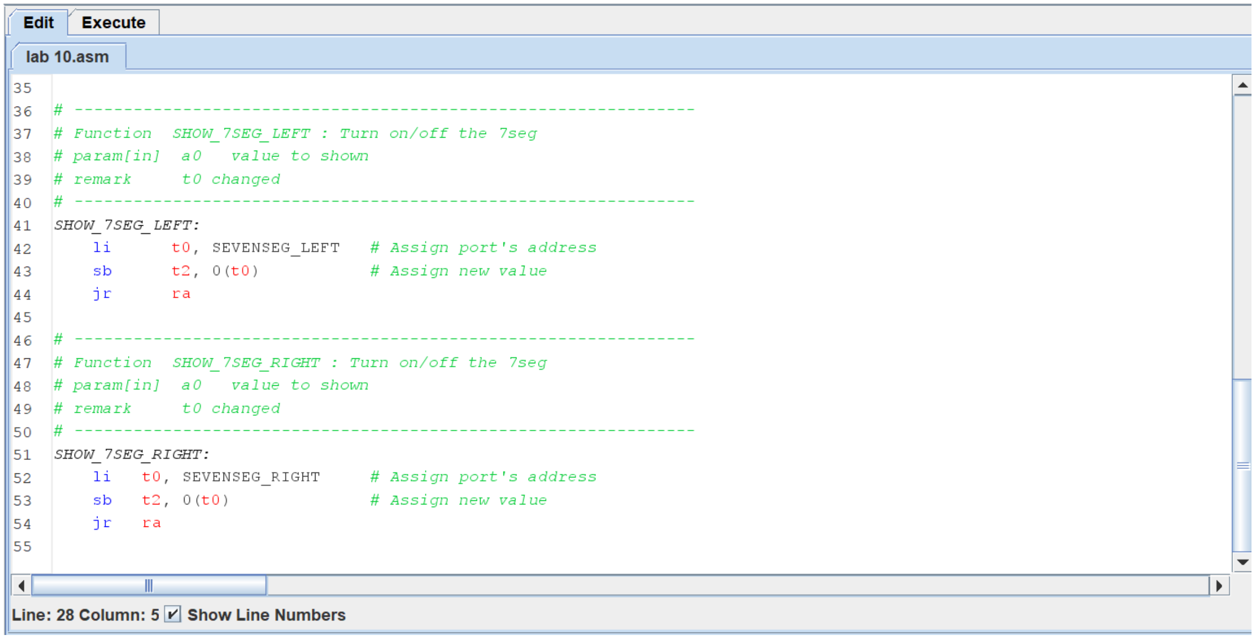
```
.eqv SEVENSEG_LEFT    0xFFFF0011    # Address of the LED on the left
                                    #     Bit 0 = segment a
                                    #     Bit 1 = segment b
                                    #     ...
                                    #     Bit 7 = dot sign
.eqv SEVENSEG_RIGHT   0xFFFF0010    # Address of the LED on the right
.data
        char: .half
        num: .byte 0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f
.text
main: # present the last two digits of ASCII code of the character entered
    li s1, 100
    li s0, 10
    li a7, 12
    ecall
    rem a0, a0, s1

    div s2, a0, s0
    la t1, num
    add t1, t1, s2

    lb      t2,  0(t1)              # Set value for 7 segments to present the first digit
```

Line: 28 Column: 5 ☑ Show Line Numbers



```
    add t1, t1, s2

    lb      t2,  0(t1)              # Set value for 7 segments to present the first digit
    jal     SHOW_7SEG_LEFT         # Show the result

    rem s2, a0, s0
    la t1, num
    add t1, t1, s2

    lb      t2,  0(t1)             # Set value for 7 segments to present the second digit
    jal     SHOW_7SEG_RIGHT        # Show the result
exit:
    li      a7, 10
    ecall
end_main:

# ---------------------------------------------------------------
# Function  SHOW_7SEG_LEFT : Turn on/off the 7seg
# param[in]  a0   value to shown
# remark      t0 changed
# ---------------------------------------------------------------
SHOW_7SEG_LEFT:
```

Line: 28 Column: 5 ☑ Show Line Numbers



```
# --------------------------------------------------------------
# Function  SHOW_7SEG_LEFT : Turn on/off the 7seg
# param[in]  a0   value to shown
# remark      t0 changed
# --------------------------------------------------------------
SHOW_7SEG_LEFT:
    li      t0, SEVENSEG_LEFT    # Assign port's address
    sb      t2, 0(t0)            # Assign new value
    jr      ra

# --------------------------------------------------------------
# Function  SHOW_7SEG_RIGHT : Turn on/off the 7seg
# param[in]  a0   value to shown
# remark      t0 changed
# --------------------------------------------------------------
SHOW_7SEG_RIGHT:
    li   t0, SEVENSEG_RIGHT      # Assign port's address
    sb   t2, 0(t0)               # Assign new value
    jr   ra
```

Line: 28 Column: 5 ☑ Show Line Numbers

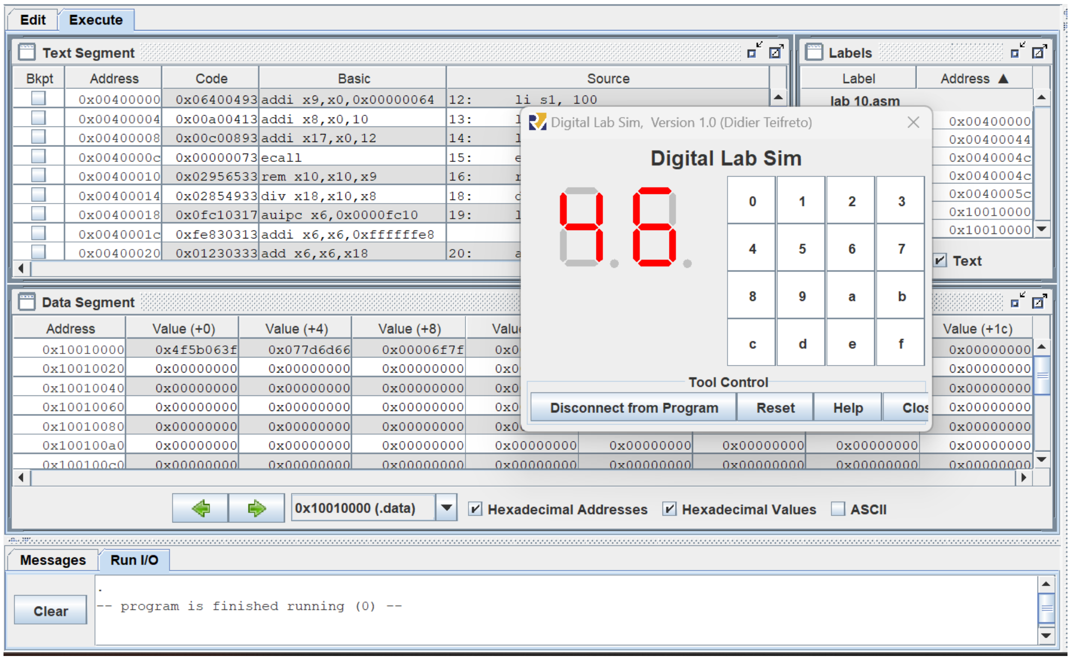
I create an array num containing the value of seven segment display of the digit from 0 to 9. Num[i] will display the digit i. Then, consider the last second digit and the last digit. Register t1 stores address of array num.

⇨ (t1 + i) stores the value of seven segment display to display digit i.

Hence, we can easily present the last two digits of the ASCII code of a character entered from the keyboard.

For some examples:

- If we input ‘ . ’, the result is:



Text Segment

Bkpt	Address	Code	Basic	Source
□	0x00400000	0x06400493	addi x9,x0,0x00000064	12: li s1, 100
□	0x00400004	0x00a00413	addi x8,x0,10	13: l
□	0x00400008	0x00c00893	addi x17,x0,12	14: l
□	0x0040000c	0x00000073	ecall	15: e
□	0x00400010	0x02956533	rem x10,x10,x9	16: r
□	0x00400014	0x02854933	div x18,x10,x8	18: d
□	0x00400018	0x0fc10317	auipc x6,0x0000fc10	19: l
□	0x0040001c	0xfe830313	addi x6,x6,0xffffffe8	
□	0x00400020	0x01230333	add x6,x6,x18	20: a

Labels

Label	Address ▲
lab 10.asm	
	0x00400000
	0x00400044
	0x0040004c
	0x0040004c
	0x0040005c
	0x10010000
	0x10010000

☑ Text

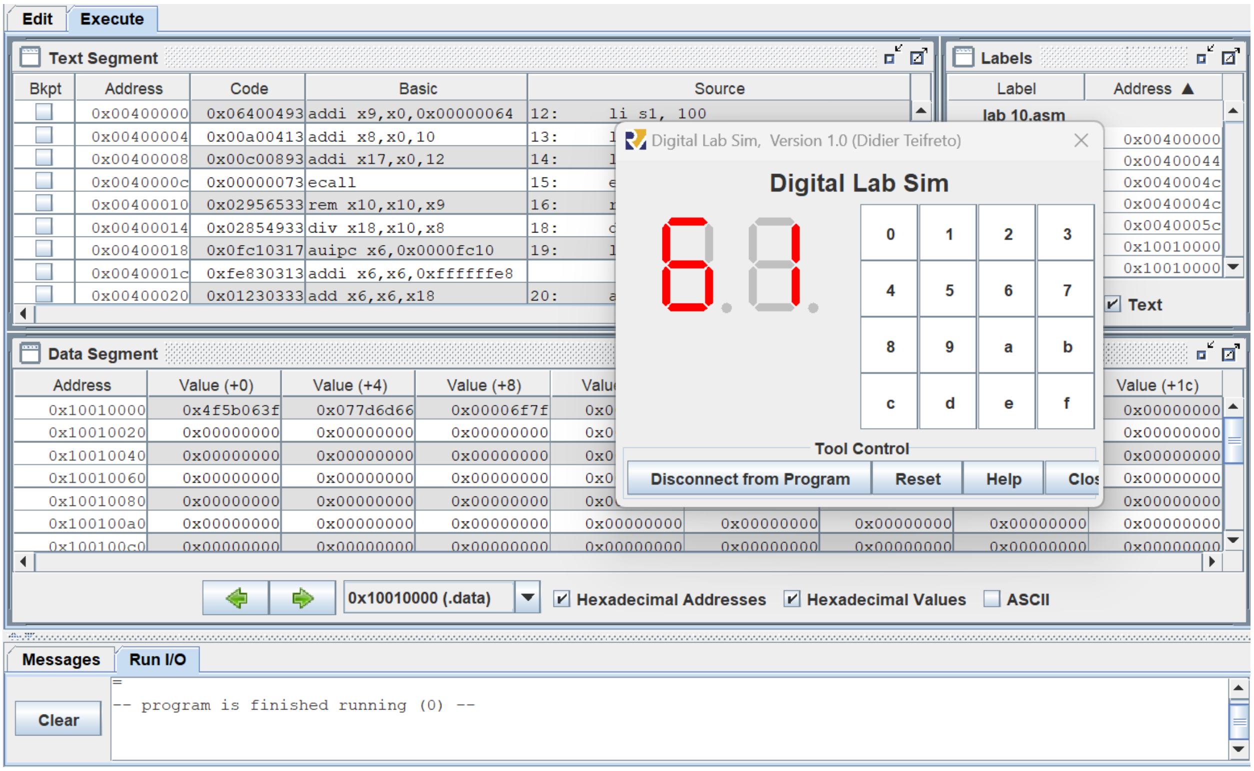
Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Valu				Value (+1c)
0x10010000	0x4f5b063f	0x077d6d66	0x00006f7f	0x0				0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values □ ASCII

```
.
-- program is finished running (0) --
```

- If we input '=', the result is:



Bkpt	Address	Code	Basic	Source
□	0x00400000	0x06400493	addi x9,x0,0x00000064	12: li s1, 100
□	0x00400004	0x00a00413	addi x8,x0,10	13: l
□	0x00400008	0x00c00893	addi x17,x0,12	14: l
□	0x0040000c	0x00000073	ecall	15: e
□	0x00400010	0x02956533	rem x10,x10,x9	16: r
□	0x00400014	0x02854933	div x18,x10,x8	18: d
□	0x00400018	0x0fc10317	auipc x6,0x0000fc10	19: l
□	0x0040001c	0xfe830313	addi x6,x6,0xffffffe8	
□	0x00400020	0x01230333	add x6,x6,x18	20: a

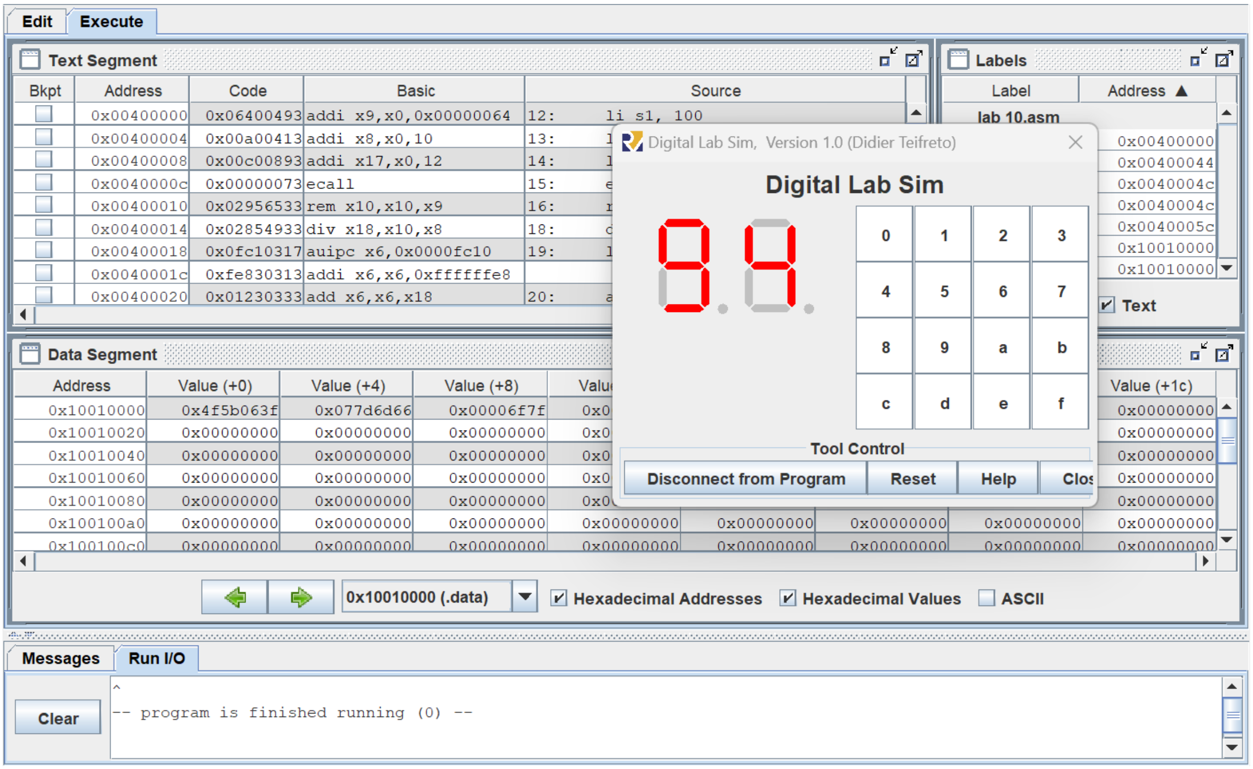
Label	Address ▲
lab 10.asm	
	0x00400000
	0x00400044
	0x0040004c
	0x0040004c
	0x0040005c
	0x10010000
	0x10010000

☑ Text

Address	Value (+0)	Value (+4)	Value (+8)	Valu				Value (+1c)
0x10010000	0x4f5b063f	0x077d6d66	0x00006f7f	0x0				0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

```
=
-- program is finished running (0) --
```

- If we input '^' , the result is:



Bkpt	Address	Code	Basic	Source
□	0x00400000	0x06400493	addi x9,x0,0x00000064	12: li s1, 100
□	0x00400004	0x00a00413	addi x8,x0,10	13: l
□	0x00400008	0x00c00893	addi x17,x0,12	14: l
□	0x0040000c	0x00000073	ecall	15: e
□	0x00400010	0x02956533	rem x10,x10,x9	16: r
□	0x00400014	0x02854933	div x18,x10,x8	18: d
□	0x00400018	0x0fc10317	auipc x6,0x0000fc10	19: l
□	0x0040001c	0xfe830313	addi x6,x6,0xffffffe8	
□	0x00400020	0x01230333	add x6,x6,x18	20: a

Label	Address ▲
lab 10.asm	
	0x00400000
	0x00400044
	0x0040004c
	0x0040004c
	0x0040005c
	0x10010000
	0x10010000

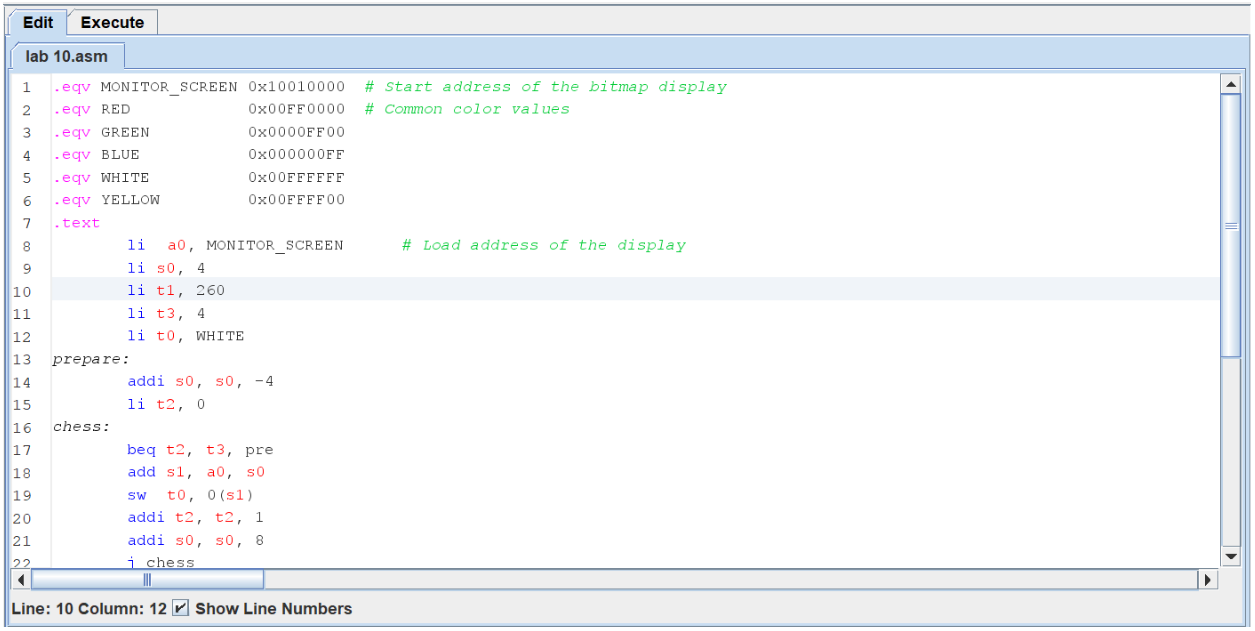
☑ Text

Address	Value (+0)	Value (+4)	Value (+8)	Valu				Value (+1c)
0x10010000	0x4f5b063f	0x077d6d66	0x00006f7f	0x0				0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x0				0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

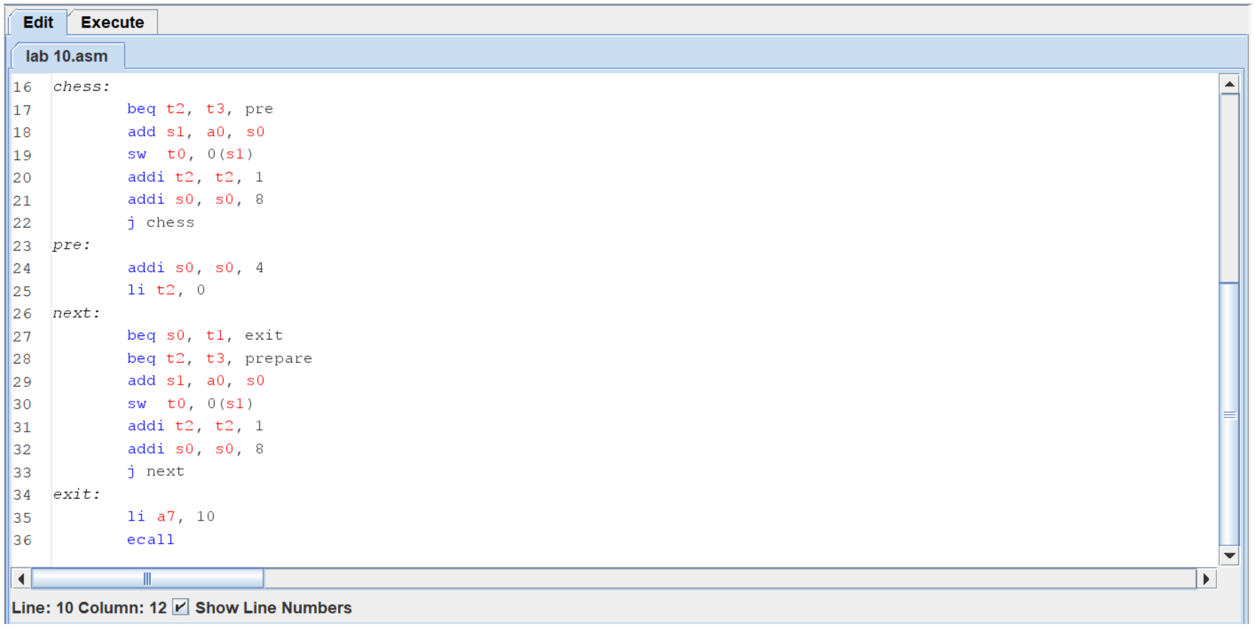
```
^
-- program is finished running (0) --
```

Assignment 3: Implement the program in Home Assignment 2, and then update the code so that it can draw a chess board.

The source code is as below:



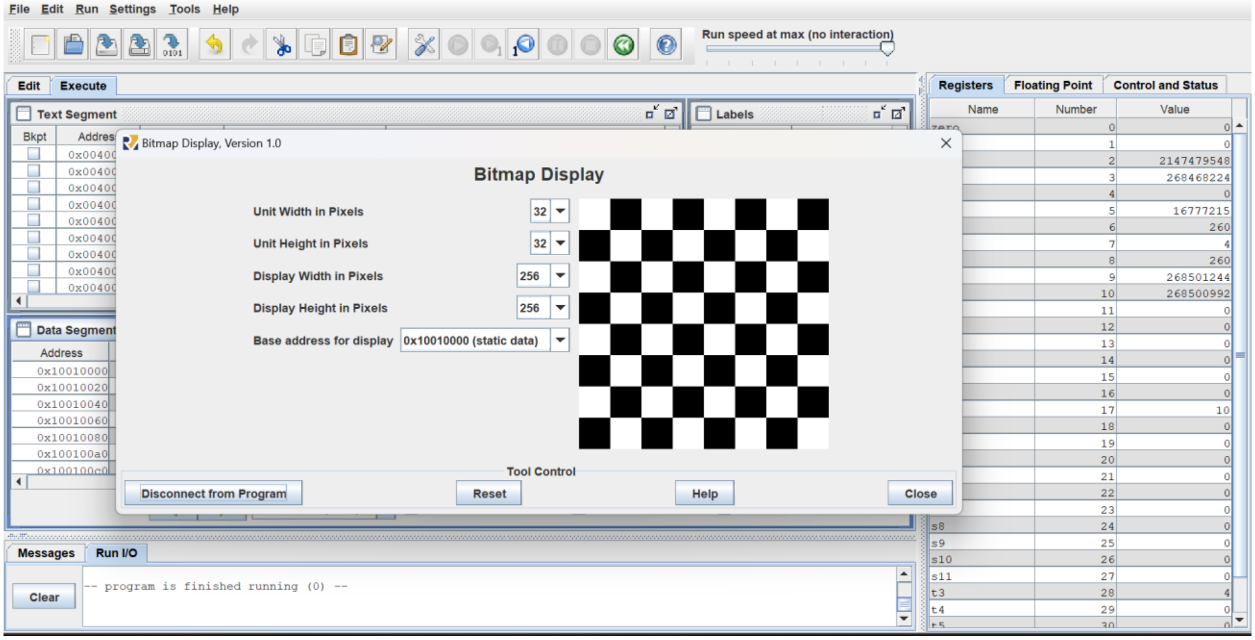
```
.eqv MONITOR_SCREEN 0x10010000  # Start address of the bitmap display
.eqv RED            0x00FF0000  # Common color values
.eqv GREEN          0x0000FF00
.eqv BLUE           0x000000FF
.eqv WHITE          0x00FFFFFF
.eqv YELLOW         0x00FFFF00
.text
        li  a0, MONITOR_SCREEN      # Load address of the display
        li s0, 4
        li t1, 260
        li t3, 4
        li t0, WHITE
prepare:
        addi s0, s0, -4
        li t2, 0
chess:
        beq t2, t3, pre
        add s1, a0, s0
        sw  t0, 0(s1)
        addi t2, t2, 1
        addi s0, s0, 8
        j chess
```



```
chess:
        beq t2, t3, pre
        add s1, a0, s0
        sw  t0, 0(s1)
        addi t2, t2, 1
        addi s0, s0, 8
        j chess
pre:
        addi s0, s0, 4
        li t2, 0
next:
        beq s0, t1, exit
        beq t2, t3, prepare
        add s1, a0, s0
        sw  t0, 0(s1)
        addi t2, t2, 1
        addi s0, s0, 8
        j next
exit:
        li a7, 10
        ecall
```

We calculate the address of white squares and load color to it. In odd row, white starts with the first square and starts with the second square in even row. There are 8 bytes distance of their address of two white squares in a row. Difference of the address of the last white square in odd row and the first white square in even row are 12 bytes. Difference of the address of the last white square in even row and the first white square in odd row are 4 bytes.

Finally, we get the result as below:



Unit Width in Pixels: 32
Unit Height in Pixels: 32
Display Width in Pixels: 256
Display Height in Pixels: 256
Base address for display: 0x10010000 (static data)

Name	Number	Value
zero	0	0
	1	0
	2	2147479548
	3	268468224
	4	0
	5	16777215
	6	260
	7	4
	8	260
	9	268501244
	10	268500992
	11	0
	12	0
	13	0
	14	0
	15	0
	16	0
	17	10
	18	0
	19	0
	20	0
	21	0
	22	0
	23	0
s8	24	0
s9	25	0
s10	26	0
s11	27	0
t3	28	4
t4	29	0
t5	30	0

```
-- program is finished running (0) --
```

Assignment 4: Implement the program in Home Assignment 3, then update the code so that it can be executed as follows:

Enter a lowercase character => Display the corresponding uppercase character.

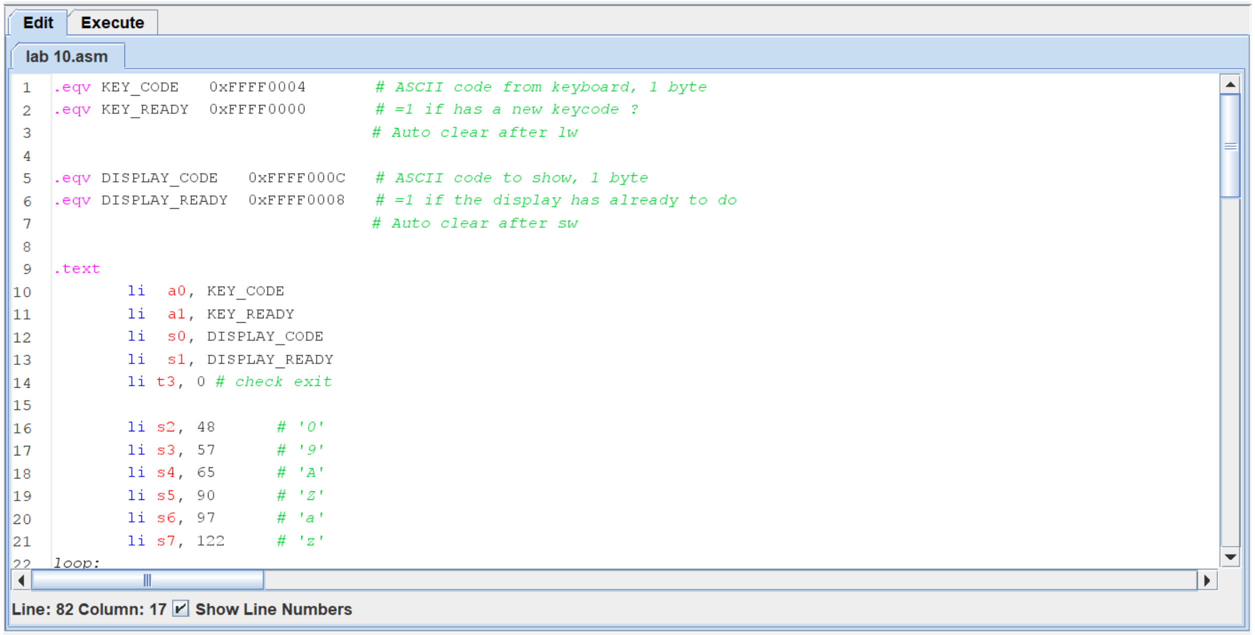
Enter an uppercase character => Display the corresponding lowercase character.

Enter a digit => Display the same digit

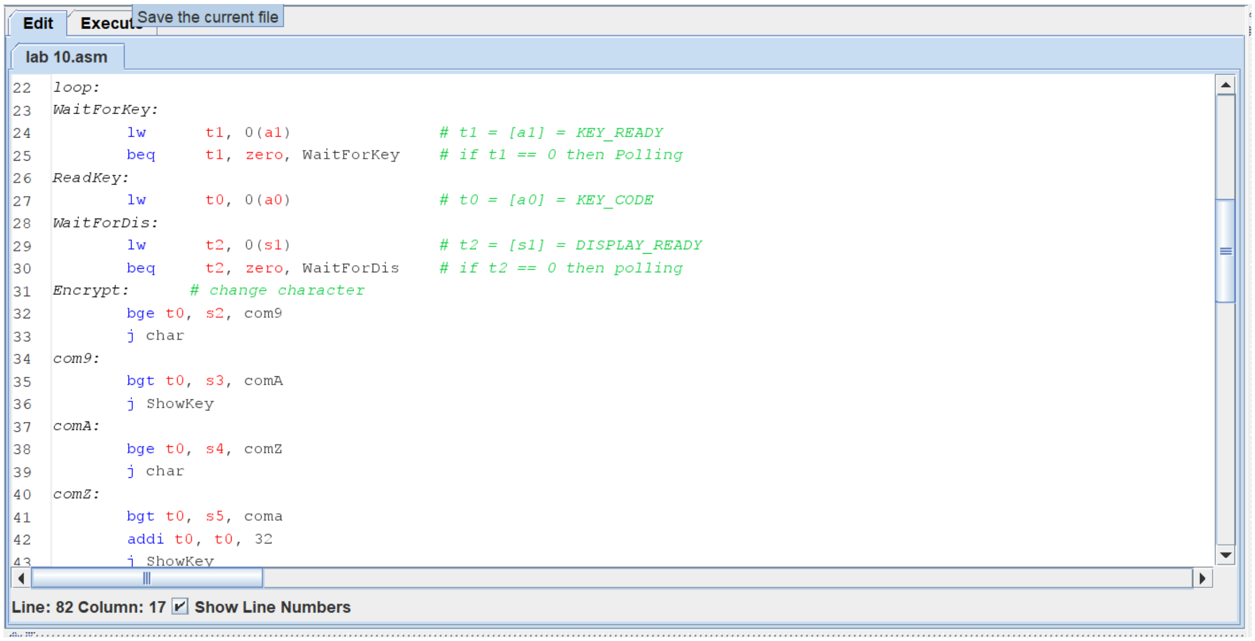
Enter another character => Display "*"

The program will be exited if "exit" is entered.

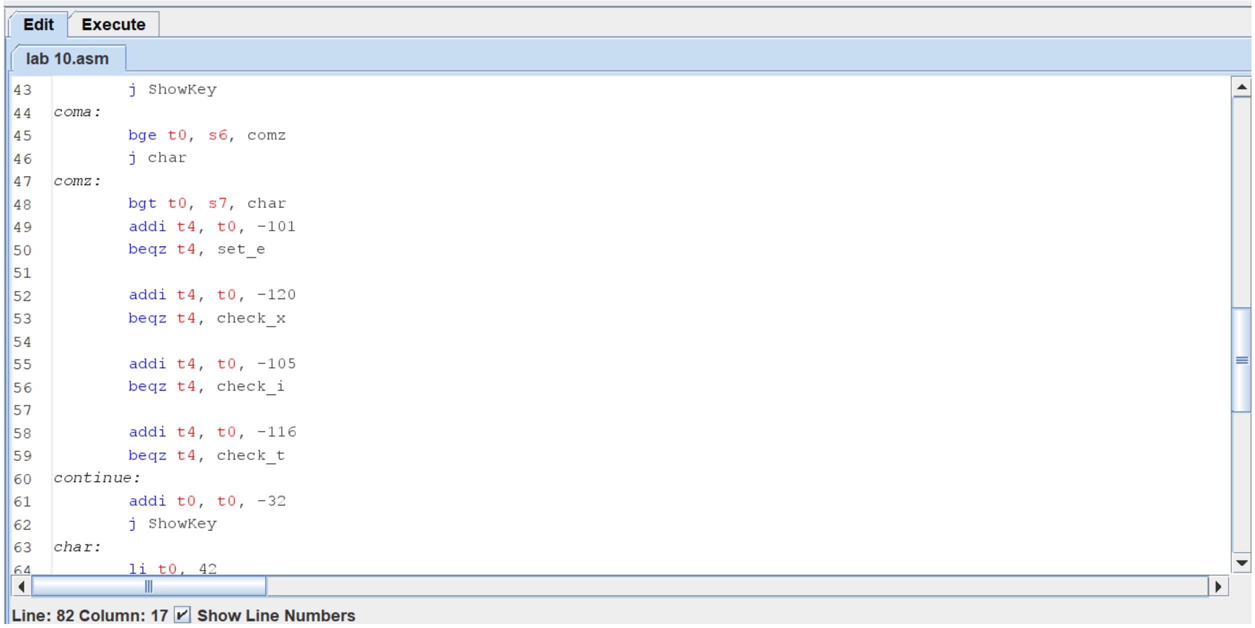
The source code is as below:



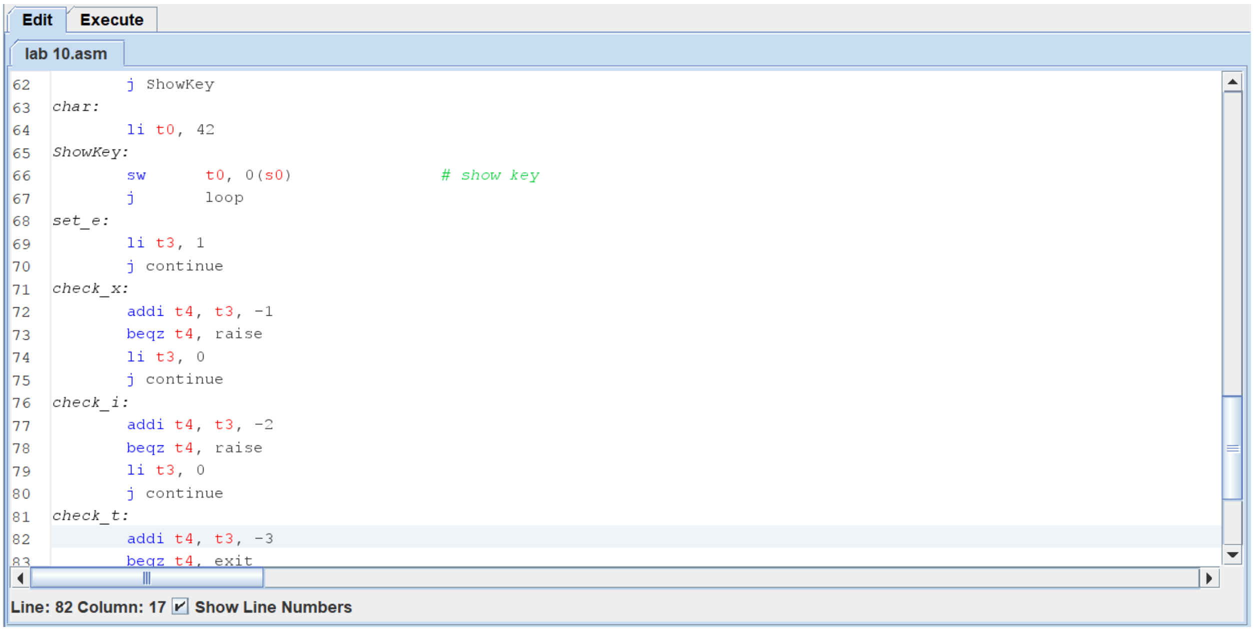
```
.eqv KEY_CODE    0xFFFF0004        # ASCII code from keyboard, 1 byte
.eqv KEY_READY   0xFFFF0000        # =1 if has a new keycode ?
                                   # Auto clear after lw

.eqv DISPLAY_CODE    0xFFFF000C    # ASCII code to show, 1 byte
.eqv DISPLAY_READY   0xFFFF0008    # =1 if the display has already to do
                                   # Auto clear after sw

.text
        li   a0, KEY_CODE
        li   a1, KEY_READY
        li   s0, DISPLAY_CODE
        li   s1, DISPLAY_READY
        li t3, 0 # check exit

        li s2, 48       # '0'
        li s3, 57       # '9'
        li s4, 65       # 'A'
        li s5, 90       # 'Z'
        li s6, 97       # 'a'
        li s7, 122      # 'z'
loop:
```



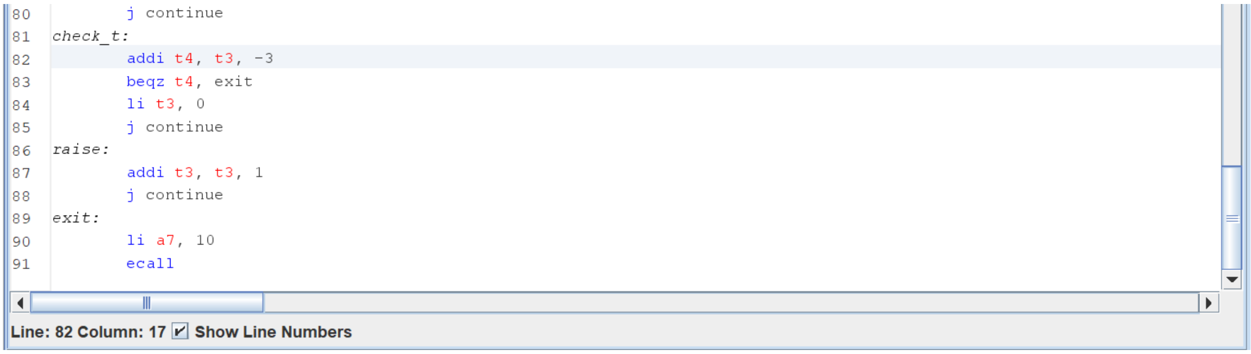
```
loop:
WaitForKey:
        lw      t1, 0(a1)               # t1 = [a1] = KEY_READY
        beq     t1, zero, WaitForKey    # if t1 == 0 then Polling
ReadKey:
        lw      t0, 0(a0)               # t0 = [a0] = KEY_CODE
WaitForDis:
        lw      t2, 0(s1)               # t2 = [s1] = DISPLAY_READY
        beq     t2, zero, WaitForDis    # if t2 == 0 then polling
Encrypt:      # change character
        bge t0, s2, com9
        j char
com9:
        bgt t0, s3, comA
        j ShowKey
comA:
        bge t0, s4, comZ
        j char
comZ:
        bgt t0, s5, coma
        addi t0, t0, 32
        j ShowKey
```



```
        j ShowKey
coma:
        bge t0, s6, comz
        j char
comz:
        bgt t0, s7, char
        addi t4, t0, -101
        beqz t4, set_e

        addi t4, t0, -120
        beqz t4, check_x

        addi t4, t0, -105
        beqz t4, check_i

        addi t4, t0, -116
        beqz t4, check_t
continue:
        addi t0, t0, -32
        j ShowKey
char:
        li t0, 42
```



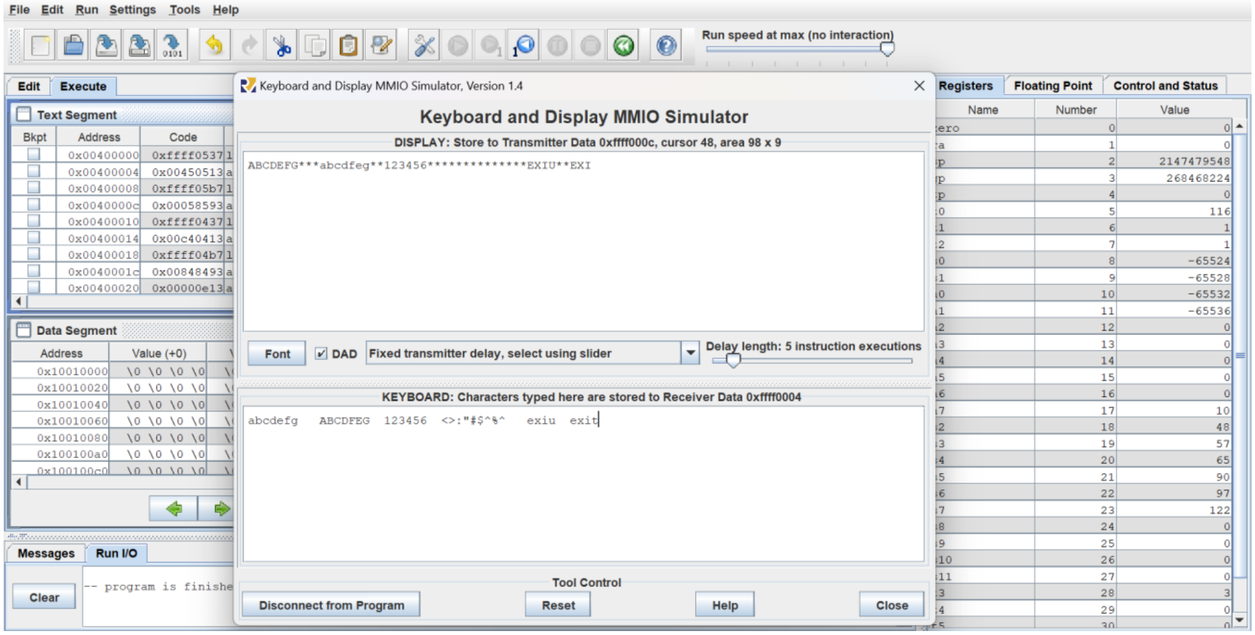
```
        j ShowKey
char:
        li t0, 42
ShowKey:
        sw      t0, 0(s0)               # show key
        j       loop
set_e:
        li t3, 1
        j continue
check_x:
        addi t4, t3, -1
        beqz t4, raise
        li t3, 0
        j continue
check_i:
        addi t4, t3, -2
        beqz t4, raise
        li t3, 0
        j continue
check_t:
        addi t4, t3, -3
        beqz t4, exit
```



```
        j continue
check_t:
        addi t4, t3, -3
        beqz t4, exit
        li t3, 0
        j continue
raise:
        addi t3, t3, 1
        j continue
exit:
        li a7, 10
        ecall
```

We mark if e, x, i, t appear respectively. If mark storing in t3 is 3, then end the program

The result is as below:


File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Keyboard and Display MMIO Simulator, Version 1.4
Registers Floating Point Control and Status
Font
☑ DAD
Fixed transmitter delay, select using slider
Delay length: 5 instruction executions
Tool Control
Disconnect from Program
Reset
Help
Close
Messages Run I/O
Clear

Text Segment

Bkpt	Address	Code
□	0x00400000	0xffff0537 l
□	0x00400004	0x00450513 a
□	0x00400008	0xffff05b7 l
□	0x0040000c	0x00058593 a
□	0x00400010	0xffff0437 l
□	0x00400014	0x00c40413 a
□	0x00400018	0xffff04b7 l
□	0x0040001c	0x00848493 a
□	0x00400020	0x00000e13 a

Data Segment

Address	Value (+0)
0x10010000	\0 \0 \0 \0
0x10010020	\0 \0 \0 \0
0x10010040	\0 \0 \0 \0
0x10010060	\0 \0 \0 \0
0x10010080	\0 \0 \0 \0
0x100100a0	\0 \0 \0 \0
0x100100c0	\0 \0 \0 \0

```
-- program is finishe
```

Keyboard and Display MMIO Simulator

DISPLAY: Store to Transmitter Data 0xffff000c, cursor 48, area 98 x 9

```
ABCDEFG***abcdfeg**123456*************EXIU**EXI
```

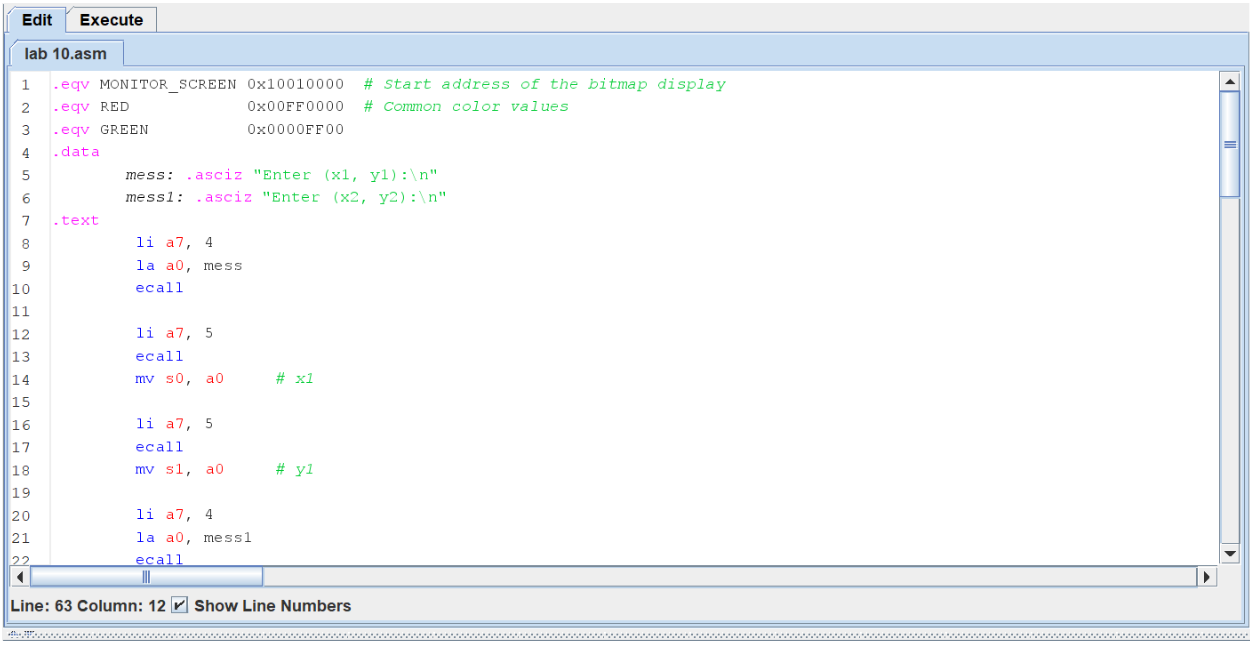
KEYBOARD: Characters typed here are stored to Receiver Data 0xffff0004

```
abcdefg   ABCDFEG  123456  <>:"#$^%^   exiu  exit
```

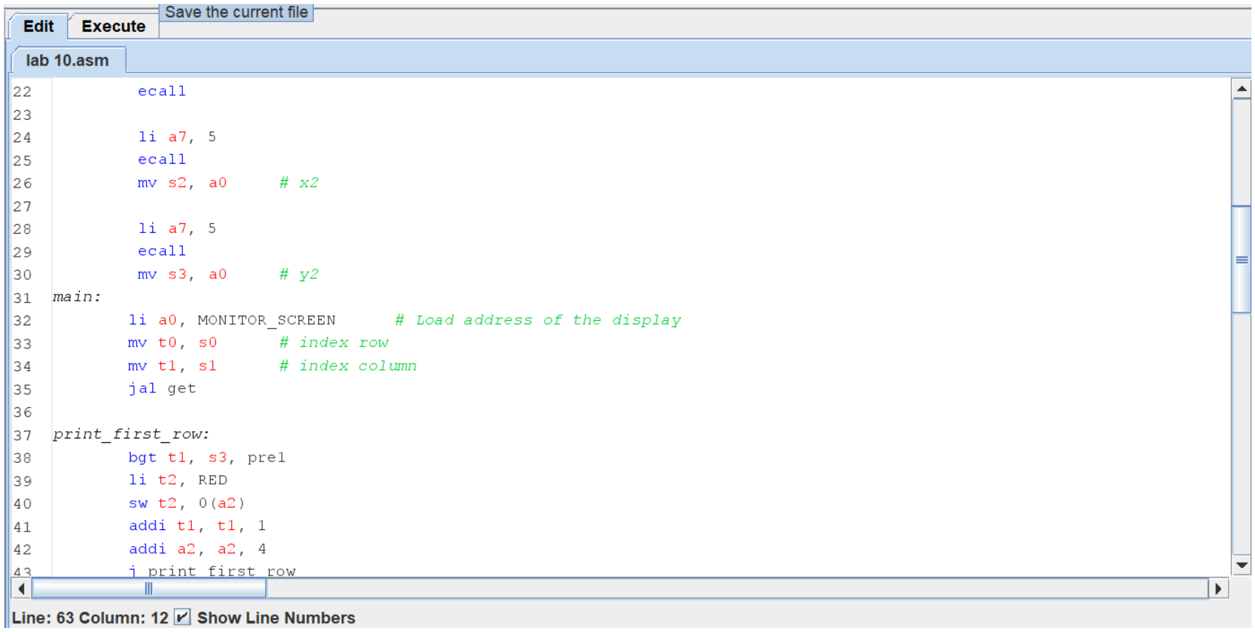
Name	Number	Value
ero	0	0
a	1	0
p	2	2147479548
p	3	268468224
p	4	0
0	5	116
1	6	1
2	7	1
0	8	-65524
1	9	-65528
0	10	-65532
1	11	-65536
2	12	0
3	13	0
4	14	0
5	15	0
6	16	0
7	17	10
2	18	48
3	19	57
4	20	65
5	21	90
6	22	97
7	23	122
8	24	0
9	25	0
10	26	0
11	27	0
3	28	3
4	29	0
t5	30	0

Assignment 5: Write a program that allows the user to enter 2 points with coordinates (x1, y1) and (x2, y2) (x1 is different from x2 and y1 is different from y2), draw and color a rectangle with 2 corners being the 2 entered points with a red border 1 unit wide and a green background. For example, with (x1, y1) = (3, 3) and (x2, y2) = (18, 11), or (x1, y1) = (3, 11) and (x2, y2) = (18, 3), we will have the result as the following figure.

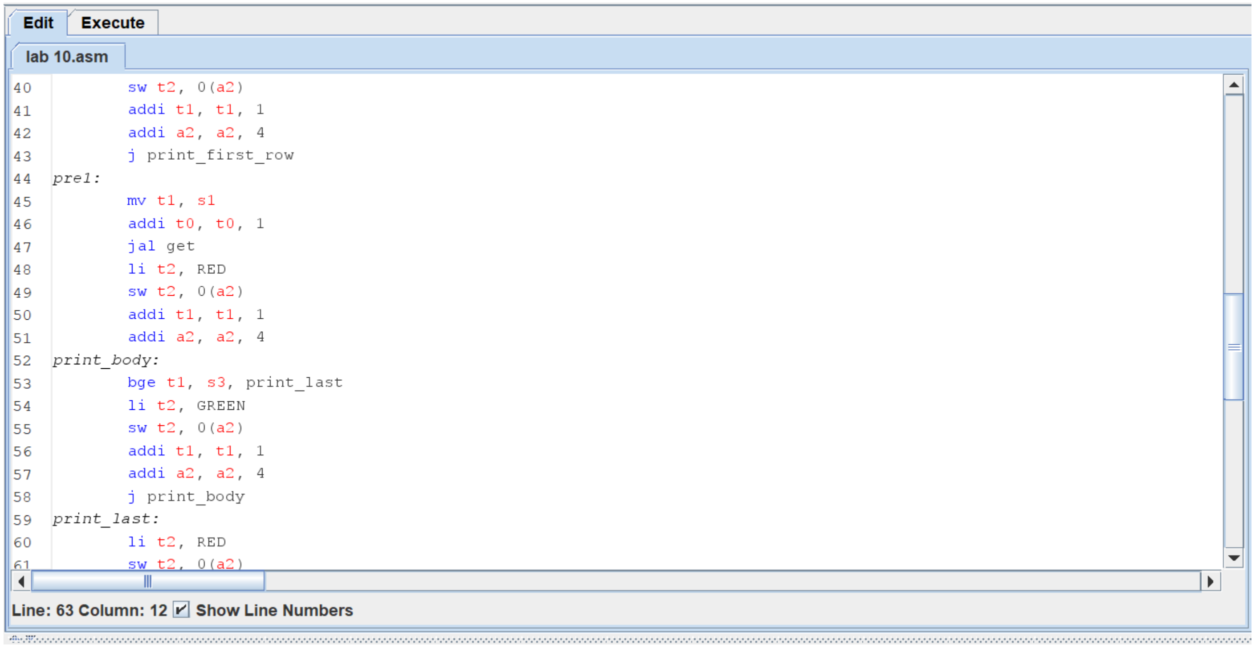
The source code is as below:



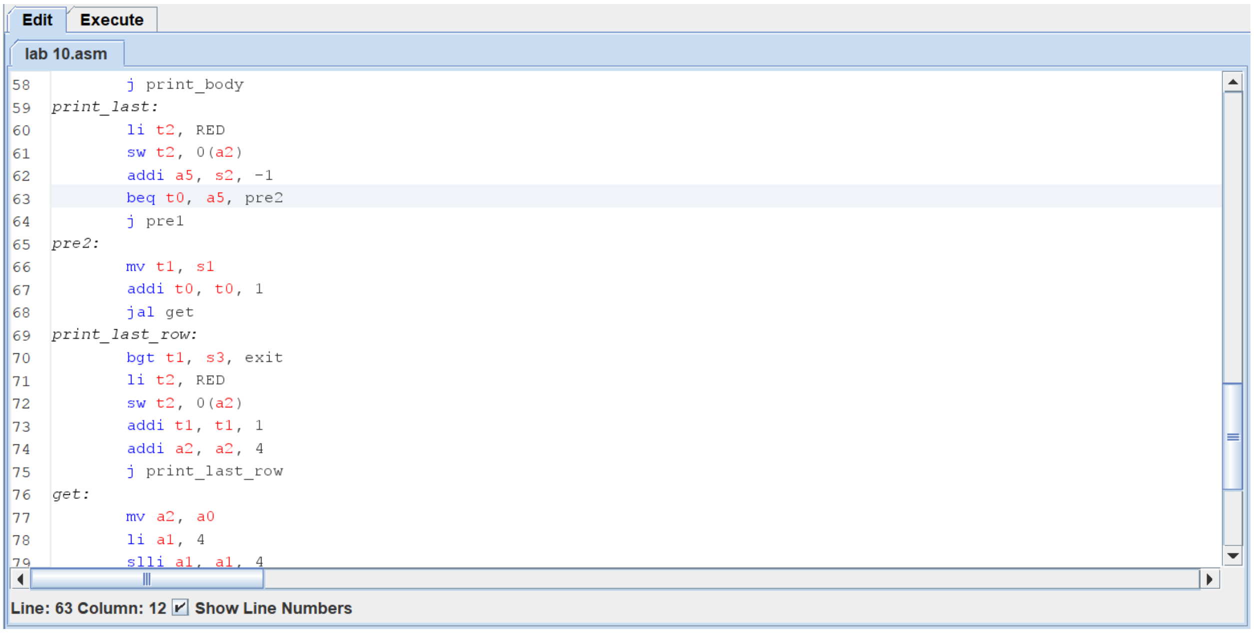
```
.eqv MONITOR_SCREEN 0x10010000  # Start address of the bitmap display
.eqv RED            0x00FF0000  # Common color values
.eqv GREEN          0x0000FF00
.data
        mess: .asciz "Enter (x1, y1):\n"
        mess1: .asciz "Enter (x2, y2):\n"
.text
         li a7, 4
         la a0, mess
         ecall

         li a7, 5
         ecall
         mv s0, a0      # x1

         li a7, 5
         ecall
         mv s1, a0      # y1

         li a7, 4
         la a0, mess1
         ecall
```



```
         ecall

         li a7, 5
         ecall
         mv s2, a0      # x2

         li a7, 5
         ecall
         mv s3, a0      # y2
main:
        li a0, MONITOR_SCREEN       # Load address of the display
        mv t0, s0       # index row
        mv t1, s1       # index column
        jal get

print_first_row:
        bgt t1, s3, pre1
        li t2, RED
        sw t2, 0(a2)
        addi t1, t1, 1
        addi a2, a2, 4
        j print_first_row
```

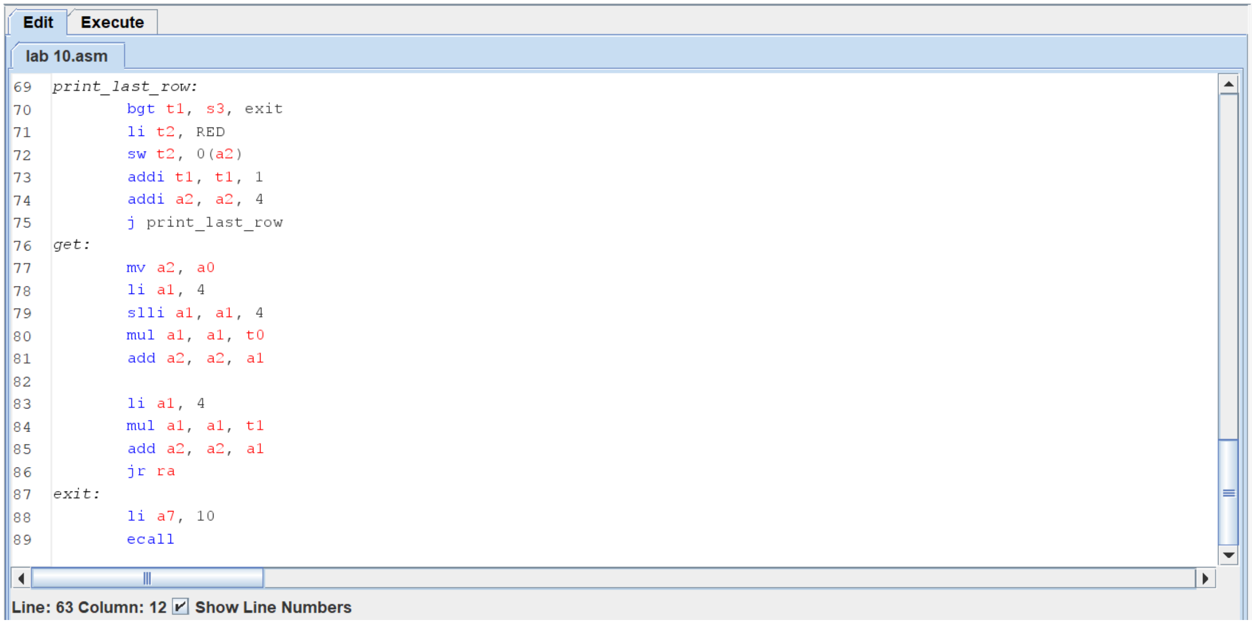


```
        sw t2, 0(a2)
        addi t1, t1, 1
        addi a2, a2, 4
        j print_first_row
pre1:
        mv t1, s1
        addi t0, t0, 1
        jal get
        li t2, RED
        sw t2, 0(a2)
        addi t1, t1, 1
        addi a2, a2, 4
print_body:
        bge t1, s3, print_last
        li t2, GREEN
        sw t2, 0(a2)
        addi t1, t1, 1
        addi a2, a2, 4
        j print_body
print_last:
        li t2, RED
        sw t2, 0(a2)
```



```
        j print_body
print_last:
        li t2, RED
        sw t2, 0(a2)
        addi a5, s2, -1
        beq t0, a5, pre2
        j pre1
pre2:
        mv t1, s1
        addi t0, t0, 1
        jal get
print_last_row:
        bgt t1, s3, exit
        li t2, RED
        sw t2, 0(a2)
        addi t1, t1, 1
        addi a2, a2, 4
        j print_last_row
get:
        mv a2, a0
        li a1, 4
        slli a1, a1, 4
```

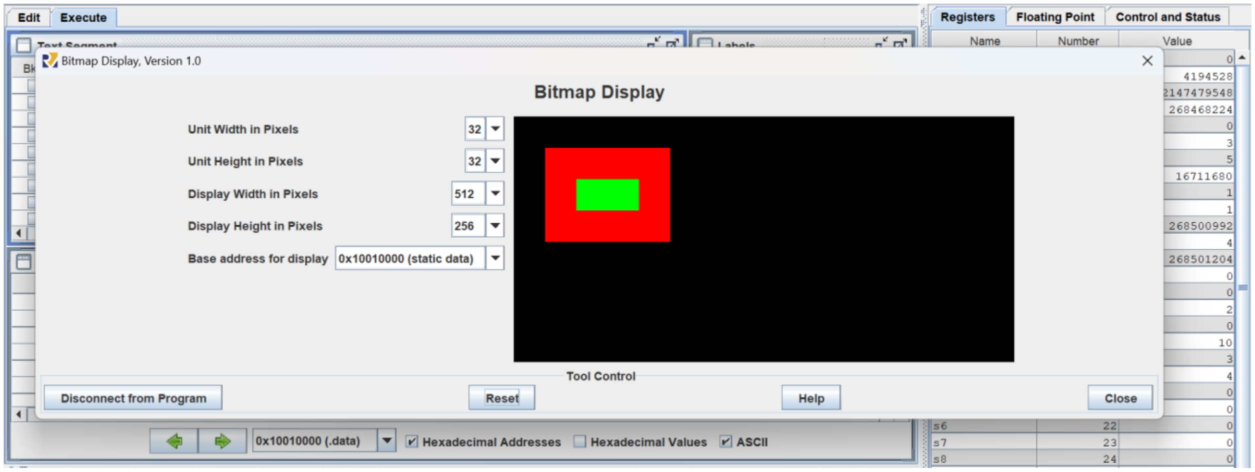
Line: 63 Column: 12 ☑ Show Line Numbers



```
print_last_row:
        bgt t1, s3, exit
        li t2, RED
        sw t2, 0(a2)
        addi t1, t1, 1
        addi a2, a2, 4
        j print_last_row
get:
        mv a2, a0
        li a1, 4
        slli a1, a1, 4
        mul a1, a1, t0
        add a2, a2, a1

        li a1, 4
        mul a1, a1, t1
        add a2, a2, a1
        jr ra
exit:
        li a7, 10
        ecall
```

Line: 63 Column: 12 ☑ Show Line Numbers

- If we input (x1, y1) = (1, 1) and (x2, y2) = (3, 4), the result is:



Unit Width in Pixels: 32

Unit Height in Pixels: 32

Display Width in Pixels: 512

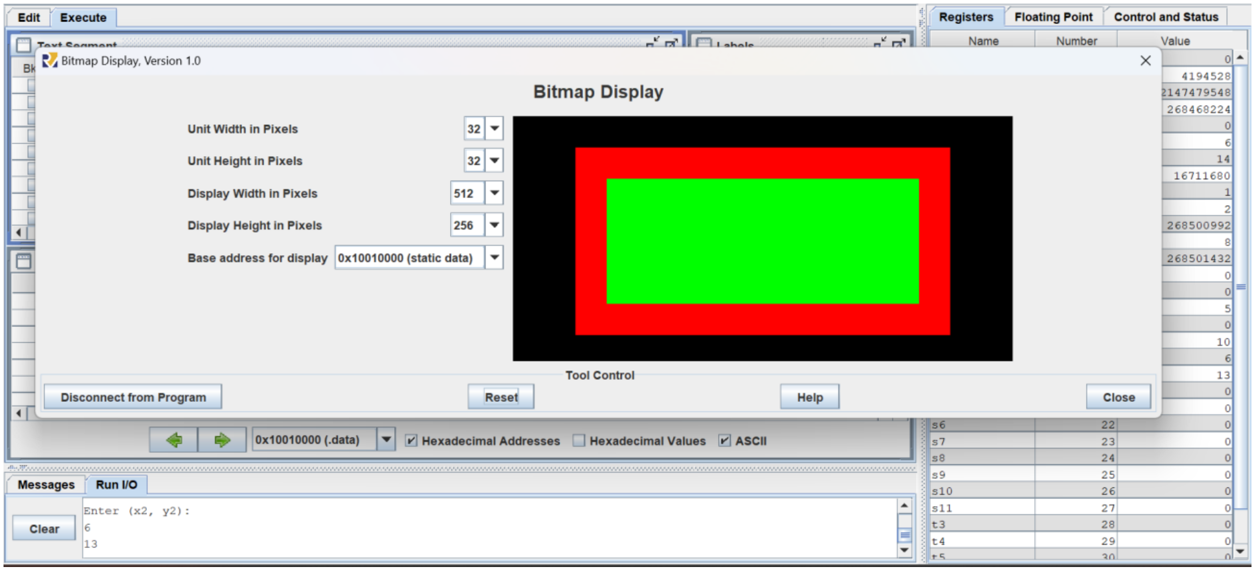
Display Height in Pixels: 256

Base address for display: 0x10010000 (static data)

0x10010000 (.data) ☑ Hexadecimal Addresses □ Hexadecimal Values ☑ ASCII

Name	Number	Value
		0
		4194528
		2147479548
		268468224
		0
		3
		5
		16711680
		1
		1
		268500992
		4
		268501204
		0
		0
		2
		0
		10
		3
		4
		0
		0
s6	22	0
s7	23	0
s8	24	0

- If we input (x1, y1) = (1, 2) and (x2, y2) = (6, 13), the result is:



Unit Width in Pixels: 32

Unit Height in Pixels: 32

Display Width in Pixels: 512

Display Height in Pixels: 256

Base address for display: 0x10010000 (static data)

0x10010000 (.data) ☑ Hexadecimal Addresses □ Hexadecimal Values ☑ ASCII

Name	Number	Value
		0
		4194528
		2147479548
		268468224
		0
		6
		14
		16711680
		1
		2
		268500992
		8
		268501432
		0
		0
		5
		0
		10
		6
		13
		0
		0
s6	22	0
s7	23	0
s8	24	0
s9	25	0
s10	26	0
s11	27	0
t3	28	0
t4	29	0
t5	30	0

```
Enter (x2, y2):
6
13
```